

## PATENT ABSTRACTS OF JAPAN

(11)Publication number : 04-051367

(43)Date of publication of application : 19.02.1992

(51)Int.Cl.

G06F 15/60

(21)Application number : 02-159948

(71)Applicant : TOSHIBA CORP

(22)Date of filing : 20.06.1990

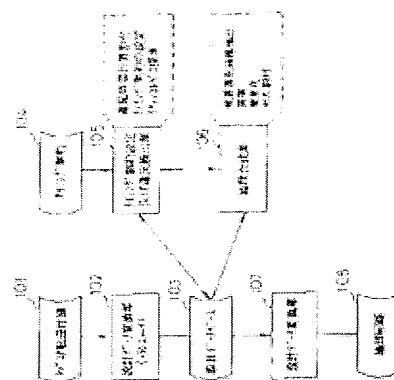
(72)Inventor : NISHI HIROAKI

## (54) SYNTHESIS DEVICE FOR LOGICAL CIRCUIT

(57)Abstract:

PURPOSE: To synthesize the logical circuits of high performance by providing a means which gives arbitrary delay time to the constitution element of an abstraction circuit, executing timing analysis at the stage of the abstraction circuit, and executing optimization so that the restriction of delay time is satisfied through the use of an analysis result.

CONSTITUTION: A timing restriction 104 sets restriction delay time in the arbitrary point of a register transfer specification. A timing restriction setting/ violation detection part 105 fetches data on the type of the element of an initial abstraction circuit from a design data base part 103, calculates the delay time of the element and allocates it to the initial abstraction circuit generated in a design data conversion part 102, fetches the timing restriction 104 and inspects whether given timing restriction is realized or not. A logical synthesis part 106 analyzes a violation circuit detected in the violation detection part 105, develops the abstraction circuit to a concrete circuit, simplifies a redundancy circuit and allocates data to the circuit element.



⑨ 日本国特許庁(JP)

⑩ 特許出願公開

⑫ 公開特許公報(A)

平4-51367

⑤ Int.Cl.<sup>5</sup>

G 06 F 15/60

識別記号

3 6 0 K

庁内整理番号

7922-5L

⑬ 公開 平成4年(1992)2月19日

審査請求 未請求 請求項の数 2 (全 17 頁)

⑭ 発明の名称 論理回路の合成装置

⑮ 特 願 平2-159948

⑯ 出 願 平2(1990)6月20日

⑰ 発 明 者 西 宏 晃 神奈川県川崎市幸区小向東芝町1 株式会社東芝総合研究  
所内

⑱ 出 願 人 株 式 会 社 東 芝 神奈川県川崎市幸区堀川町72番地

⑲ 代 理 人 弁理士 三好 秀和 外1名

明 細 書

1. 発明の名称

論理回路の合成装置

2. 特許請求の範囲

(1) 論理回路を自動生成するための論理回路の  
合成装置において、

合成の開始前、あるいはその途中で生成された  
抽象回路を使用してタイミング解析を行なうため  
に、前記抽象回路を構成している素子に任意の遅  
延時間を与える手段と、

与えられた遅延時間を用いてタイミング解析を  
行い、タイミング制約条件が満たされるかどうか  
を検出する手段と、

前記検出されたタイミング解析結果を使用して  
タイミング制約条件が満たされるように前記抽象  
回路を論理回路に変換あるいは合成する手段と、

前記タイミング制約に違反する回路の遅延時間  
を最適化するために回路を変換あるいは合成する  
手段と、

を具備することを特徴とする論理回路の合成装置。

(2) 前記遅延時間最適化のための回路の変換あ  
るいは合成手段は、前記タイミング解析手段で検  
出された制約違反の回路を解析して前記回路を最  
適化対象回路に分割するとともに分割された回路  
に最適化目標値を設定し、前記最適化目標値にし  
たがって回路を変換あるいは合成する手段を有す  
ることを特徴とする請求項1記載の論理回路の合  
成装置。

3. 発明の詳細な説明

[発明の目的]

(産業上の利用分野)

この発明は論理回路を自動的に合成するため  
の装置に係り、特にタイミングの制約に基づいて  
論理回路を合成するための装置に関する。

(従来の技術)

論理合成技術の発展によって、レジスタ転送  
レベルの回路記述から構造レベルの回路を自動生  
成できるようになった。

ところが、熟練設計者が設計した論理回路と自

動合成された論理回路の性能（特に動作速度など）を比較すると、人手設計のほうが優れている。というのは、熟練設計者は、回路の動作周波数を満たすようにレジスタのクロックを決定し、クリティカルパスを絶えず考慮しながら論理回路を設計しているからである。すなわち、どの様な構造を持った回路にすれば信号遅延時間が少なくて済むかを熟知しているからである。

また設計者はタイミング解析ツールを使用することによって、論理回路の詳細な信号遅延時間の解析結果を入手できるようになったが（例えばどのパスが最長遅延もしくは最小遅延パスであるか）、これは設計者が所望する論理回路の設計を一通り終った後でないと使用できなかった。従って、論理合成後の論理回路に対してタイミング解析を行い、設計者が考えている制約に違反する箇所を入手で修正する必要があった。

一方、最近の論理合成技術では、ライブラリ化されたハードウェアコンポーネント（ライブラリセル）をゲート回路に割り当てた後で、論理合成

システムに内蔵したタイミング解析手段を用いてタイミング解析を実行し、この結果が設計者の所望するタイミング制約を満たしていない場合は、スピードの遅いハードウェアコンポーネントをスピードの速いハードウェアコンポーネントに交換し、再びタイミング解析を実行してタイミング制約が満たされているかどうかを調べ、これが満たされるまで、もしくはその限界まで繰り返すという手法をとっていた。

他方では、タイミング情報をハードウェアコンポーネントのテクノロジーに依存しないライブラリ化された抽象コンポーネントに格納し、これを合成時に使用する手法も考えられている。この技術に関しては、特開昭63-155268号公報において示されている。

しかし抽象コンポーネントは入力数が可変であったりコンポーネント自体ビット幅を有しているので、許されるビット幅に応じた膨大な数のコンポーネントをライブラリに登録する必要があり、従ってこの手法では多大な記録容量を必要として

— 3 —

いた。

（発明が解決しようとする課題）

上述したように従来技術では、部品ライブラリとして用意されたコンポーネントによって回路が構成されていないと、タイミング解析が実行できないという不具合があった。

また、レジスタ転送回路から変換された抽象回路では元来入出力数が可変のため、遅延時間を使用可能な全ての抽象回路に持たせて、それぞれをライブラリ化するとデータ量が膨大になり不経済である。

本発明は、このような点に関してなされたもので、その目的は、同一機能を有するが、しかし異なる構造を持ったハードウェアに適した遅延時間の割付手段を用いて抽象回路の段階でタイミング解析を実行し、その結果のタイミング違反情報を合成開始前あるいは途中の段階で使用することで高性能な論理回路を合成することが可能な装置を提供することである。

〔発明の構成〕

— 5 —

— 4 —

（課題を解決するための手段）

上記課題を解決するために本発明では、論理回路を自動生成するための論理回路の合成装置において、合成の開始前、あるいはその途中で生成された抽象回路を使用してタイミング解析を行なうために、前記抽象回路を構成している素子に任意の遅延時間を与える手段と、与えられた遅延時間を用いてタイミング解析を行い、タイミング制約条件が満たされるかどうかを検出する手段と、前記検出されたタイミング解析結果を使用してタイミング制約条件が満たされるように前記抽象回路を論理回路に変換あるいは合成する手段と、前記タイミング制約に違反する回路の遅延時間を最適化するために回路を変換あるいは合成する手段と、を具備することを特徴とする。

（作用）

本発明の装置によれば、機能動作仕様記述から変換された抽象回路を構成する素子に任意の遅延時間を割当てることができ、この遅延時間を使用して合成前あるいは合成途中にタイミング検証

— 6 —

が行える。さらにタイミング解析の結果を使用して遅延時間の制約を満足するように抽象回路を論理回路に変換または合成し最適化を繰返して行えるので、スピードの速い論理回路の合成ができる。

#### (実施例)

以下、本発明の実施例を図面を参照して説明する。

第1図は本発明の一実施例にかかる論理合成装置の構成を示すブロック図である。

第1図の101は機能動作仕様を表すレジスタ転送仕様の入力部で、レジスタ転送記述もしくは該仕様をスキマテック表示した機能ブロック図である。102は設計データ変換部(トランスレータ)である。この変換部102は入力部101のレジスタ転送記述を構文解析して、該記述に忠実な初期抽象回路を生成する。また機能ブロック図の場合はスキマテック図よりネットワーク情報と図形情報を抽出して初期抽象回路を生成する。103は該抽象回路のネットワーク情報を格納する設計データベース部である。104はタイミング

制約入力部で、レジスタ転送仕様の任意の点に制約遅延時間を設定できる。105はタイミング制約設定及び違反検出部で、トランスレータ部102で生成された抽象機能素子の入出力数が一定でないため、設計データベース部103から該初期抽象回路の素子のタイプと入力数またはビット幅を取り出し、該素子の遅延時間を計算し割り付ける遅延時間計算割付手段と、入力部104からタイミング制約を取り込む手段と、与えられたタイミング制約が、論理合成前及び途中の抽象回路もしくは論理回路において実現されるかどうかを検証するクリティカルパス探索手段から構成されている。106は論理合成部で、前記タイミング制約違反検出部105で検出された違反回路を解析して該回路を最適化対象回路に分割し、さらに最適化目標値を設定し、該最適化目標値に従って抽象回路を具象回路に展開し、また、冗長な回路の単純化及び、テクノロジーに依存した回路素子の割付けを行う各処理手段から構成されている。107は設計データ変換部で、記述変換やスキマテック

— 7 —

— 8 —

ク図の生成を行うためのもので、具体的には設計データベースに格納されている回路情報を論理回路(構造記述またはスキマテック図)に変換する。108は合成された論理回路の出力部である。

第2図は、第1図で説明した論理合成装置の処理フローを説明する図である。

201は、レジスタ転送記述及びタイミング制約を入力するステップで、202は、記述の動作解析を行って初期抽象回路を生成するステップである。203は抽象回路素子に遅延時間を計算して割り付けるステップである。204は、割り付けた遅延時間を使用して全体回路の遅延時間計算を行い、クリティカルパスを検出しタイミング制約が守られているかどうかを調べるステップである。制約が守られていないときは、制約に違反している箇所をレポートもしくはスキマテック表示する。また、該タイミング制約に違反しているパスに関しては、この制約が実現されるようにスピードの速い回路に展開して最適化を行い(205)、ステップ203及び204を繰返して該タイ

ミング制約が満たされているかどうかを調べ、ハードウェアコンポーネントを割り付ける(206)ことによって合成を終了する。以下各ステップの処理内容について詳細に説明する。

その前にレジスタ転送記述について説明する。レジスタ転送記述では、回路の素子と認識されるファシリティ(外部端子、内部信号、レジスタ、メモリなど)とハードウェアの詳細まで立ち入らずに機能を表せる演算子、組み込み関数(加算、減算など)、ファシリティの動作を表すプロセス(レジスタの非同期クリアなど)の記述や状態遷移表現が可能である。以下ファシリティ、演算子、組み込み関数、プロセス、ステートを抽象機能素子と呼ぶ。

ここで抽象機能素子の遅延時間について説明する。論理合成は最終的に抽象機能素子を論理素子(ゲート)まで展開して最適化を行い、これに所望のテクノロジーからなるハードウェアのライブラリセルを割り付けるものである。しかしレジスタ転送レベルの回路は機能レベルの回路であり、必

— 9 —

— 10 —

ずしもハードウェアとして最適化された回路、すなわちゲート数とかゲート段数を考慮して設計された回路とはなっていない。従って最初からすべての抽象機能素子にハードウェアのセルを割り付けて考えるのは得策ではない。また同一機能のセルであっても異なるスピードのセルを割り付けることによって実遅延時間は変わってしまうので、レジスタ転送レベルの回路の各抽象機能素子に実時間を割り当てて信号伝播パスの遅延時間を計算するのは適当ではない。そこでこの実施例の装置では、基本的なゲートを単位としたゲート段数を遅延時間の代わりに用いる。またハードウェアのNORゲートのセルなどでは入力数が増加するにしがって実遅延時間がかかなり異なってくるので（例えば6入力のNORは2入力NORの4倍の遅延時間を有する）、このようなゲートでは入力数を考慮して段数を与えることにする。

つぎに第1図の検出部105のサブ手段である遅延時間計算割付手段の処理について説明する。

まず論理合成途中で一部の抽象機能素子が具体

的なハードウェアセルに変換されてもクリティカルパスの検索が可能のように、ハードウェアセルのライブラリに存在するセルについては、それを構成する論理素子を用いて段数を計算し遅延情報のライブラリに格納する。またレジスタ転送レベルの抽象機能素子及び合成途中で生成される抽象機能素子または論理素子については、設計データベース部103から抽象機能素子を取り出して、該素子のタイプやビット幅や入力数を変数として、それらが論理合成途中に典型的な抽象素子や論理素子からなるネットワークに展開されるとした遅延モデルや、論理素子そのものの遅延モデルに従って段数を計算して、設計データベースに格納する。ここで抽象機能素子の遅延段数は抽象機能素子の展開後の回路によって異なってくるので、前記段数の計算を行う手段とともに、遅延モデルを複数用意しておく。またこの処理で一度計算した遅延段数については抽象機能素子のタイプ、入力数、ビット幅と一緒にワーク記憶領域に保存しておき、設計データベースから同一タイプの抽象機

— 1 1 —

— 1 2 —

能素子が再び取り出されたらハッシュイングを行って、該ワーク記憶領域のデータを取り出して割り付ける。

ここで第6図の10ビット2入力全加算器600を例にとって段数計算を説明する。

いま論理合成で加算器をリップルキャリー型の加算器に展開することになっている場合、第6図(b)の回路に展開されるものとして遅延モデルを考える。第6図(b)の回路は5個の2ビット2入力の全加算器601～605より構成されている。これらは下位ビットで計算されたキャリーの信号に対してシリアルに接続されており(601～605)、入力CIからCOまでのパスが最大遅延段数を有している。また出力データS<8:9>(609)も該段数と等しい遅延段数を有している。従って2ビット2入力の加算器の最大遅延段数が6段であることがわかっているとき、10ビット2入力全加算器の最大遅延段数は30段になる。

以下第3図を用いて第1図の検出部105にお

けるクリティカルパス検索手段の処理について説明する。いま、クリティカルパスの探索域は入力-レジスタ間、入力-出力間、レジスタ-レジスタ間、レジスタ-出力間とする。

処理を開始する(301)。入力属性を有するファシリティやレジスタ転送記述に於ける定数に与えた初期遅延情報や終点の信号名およびタイミング制約データ(ゲート段数)情報を取り込む(302)。レジスタ転送記述をトランスレートした結果を格納した設計データベースからネットワークデータを抽出する(303)。ネットワークを構成している機能素子が遅延情報のライブラリ内に存在するかどうかを調べ(304)、存在する場合は該ライブラリよりその遅延情報を取り込み該機能素子に割り付ける(305)。存在しない場合は展開後の回路の遅延モデルに従い段数を計算して該機能素子に割り付ける(306)。つぎに入力点より終点信号方向に向かってパスをトレースする。ここで入力点とは入力属性や定数属性を有するファシリティのことで、終点とはレジ

— 1 3 —

— 1 4 —

スタのデータ入力や出力属性のファシリティを指す。このときデータバス、制御バス（レジスタ転送記述では *if*、*case* などデータ転送を制御でき、この *if*、*case* の条件信号を作成するバス）、クロックバス（レジスタのクロック入りに接続されるバス）のそれぞれについてバスの有する最大段数を伝播しながら計算する。またクロックバスについてはレジスタに入力後はレジスタの出力に接続されているデータバスについて段数を計算する。一度計算したバス上の各素子の入力からの段数は設計データベースに格納する（307）。すべての終点に対する段数計算が終了したらつぎの処理を行う。

つぎに終点に割り付けられたタイミング制約情報を用いて、終点から入力点に向かって抽象機能素子が持っている段数を引去りながら、各抽象機能素子の出力で要求される最小の遅延段数（最もクリティカルな制約）を計算して設計データベースに格納する（308）。抽象素子間の信号はレジスタ転送レベルの回路ではファシリティとして

— 15 —

数制約に対するクリティカルバス *min*（スラック *min*）が同時に探索できる。

以下第4図を用いて第1図のクリティカルバス探索手段によって発見されたクリティカルバスから論理合成部106で行う被最適化回路（被最適化バス）の抽出と該バス上の抽象機能素子の選択、展開及び最適化処理の手順について説明する。

ここで説明の関係上本発明の論理合成が用いているルールベースの方式について簡単に説明する。レジスタ転送記述をトランスレータで初期回路に変換したあと、抽象機能素子をより抽象度の低い機能素子または論理素子に展開する。このとき展開の度合によってゲート数は多いがスピードの速い回路やスピードは遅いがゲート数は少ない回路が構成できるので、展開の制御はルールベース方式を用いている。すなわち展開の仕方をルールで選択できる。上述したように展開の仕方もルールで記述できるが展開ルールの選択制御そのものがルールで記述でき、このルールをメタルールと呼ぶことにする。また抽象機能素子同士を対象にし

— 17 —

認識されるのでバス自体の遅延についても考慮される。さらにスラック（いま求めた制約の遅延段数とバスの入力からの段数との差）を計算して設計データベースに格納する（309）。ここでスラックがマイナス値（最大遅延を考えている）になった箇所はクルティカルバス上に存在する。

その後終点からスラックの値を頼りにタイミング制約を満足しないバスをトレースする（310）。さらにこのとき制御信号の値によって機能的に非現実的なバスをクリティカルバスの候補としている場合があるので *if*、*case* の条件が矛盾なく成立するバスを本当のクリティカルバスとし、設計データベースにバス情報を格納する。そしてクリティカルバス上の各抽象素子の出力における入力からの段数、制約、スラックなどの情報を設計データベースに格納して（311）本手段の処理を終了する（312）。

またクリティカルバス探索手段では、終点に与えた最大遅延段数制約に対するクリティカルバス *max*（スラック *max*）と同じように、最小遅延段

— 16 —

た最適化はアルゴリズムを用いた手法では不可能なので、本最適化にもルールを用いている。

以下の実施例ではクリティカルバス *max* を最適化合成する手順について説明する。なお、今後特に断らない限りクリティカルバスはクリティカルバス *max* の意味で用い、スラックもスラック *max* の意味で使用する。クリティカルバスを解消するように合成するには、クリティカルバス探索手段によって計算されたスラック値をゼロにするようにすればよいのでスラック値の絶対値を最適化目標値とする。

処理を開始する（401）。クリティカルバスから被最適化バス及び最適化目標値を求める（402）。被最適化バスの選択については後述する。つぎに被最適化バス上の抽象機能素子から被展開候補抽象機能素子を搜し出し（403）（これについても後で詳しく説明する）、候補が存在する場合は、該抽象機能素子を展開したとき遅延段数が減少する（最適化目標値に近づく）かどうかを調べる（404）。なお遅延時間割付手段でレジ

— 18 —

スタ転送レベルの抽象機能素子には典型的な遅延段数を割り当ててあるので、スピードの速い抽象回路に展開するルールが存在すれば減少することがわかる。減少しない場合はつぎの候補の抽象機能素子を抽出する(403)。減少する場合は該抽象素子をより抽象度の低い抽象素子に展開する(405)。そして展開後の抽象素子に対して遅延情報ライブラリを用いて遅延段数を割り当てる。該ライブラリにない場合は計算して割り当てる(406)。展開後の抽象機能素子のネットワーク全体に対して段数計算を行い(407)、展開前の抽象機能素子の遅延段数といま計算した遅延段数を比較して、該被最適化パスの最適化目標値が達成されたかどうかを調べる(408)。達成されない場合は、つぎの候補の抽象機能素子がなくなるまで抽出して繰り返す。その後で最適化目標値が達成されない場合は、展開後の機能素子同士または被最適化パスの端点の機能素子同士について最適化を行う(409)。全ての被最適化パスに対して上記処理が終了したかどうか調べ(4

10)、終了した場合は該全体回路に対してクリティカルパス検索を行い展開最適化合成前のクリティカルパスが解消されているかどうかを調べる(411)。解消されていない場合は一連の処理でクリティカルパスが解消されずに残るまでステップ402以下の処理を繰り返す(412)。クリティカルパスが解消された場合もしくは残存するのが確定した場合は、上記展開後の抽象機能素子以外のネットワークの抽象機能素子に対しても展開を行い、さらに素子数最適化を行う(413)。このときクリティカルパスの解消のときに展開した回路については素子数最適化の処理を行わない。以上で遅延時間最適化合成の処理を終了する(414)。

つぎに第5図を用いて第4図のステップ402の被最適化パスの抽出手順について説明する。処理を開始する(501)。まずクリティカルパス探索手段で発見されたクリティカルパスmaxのスラック値から最も危険度の高い(余裕度の小さい)パスを選択する(502)。該パスが他のクリテ

— 19 —

— 20 —

ィカルパスと重複するかどうか調べ(503)、この重複パスがクリティカルパスminでないことを確かめる(504)。クリティカルパスminと重複していない場合、この重複クリティカルパスのスラックのうち危険度の最も小さい(余裕度の最も大きい)値を選ぶ(505)。そして該重複したサブパスを被最適パスとし、いま求めた危険度の最も小さい(余裕度の最も大きい)スラック値の絶対値を最適化目標値とする(506)。また非重複サブパスの最適化目標値は、各クリティカルパスが有するスラック値に重複サブパスの最適化目標値を加えた値の絶対値になる(507)。ただしこの値は重複サブパスの最適化が成功したときに用いる。すなわち最適化に失敗した場合は該重複クリティカルパスの未達最適化目標値分をそれぞれ被最適化サブパスの最適化目標値に対して加算し、最適化目標値を更新する。ステップ503で調べたクリティカルパスが他のものと重複しない場合は、取り出したクリティカルパス全体を被最適化パスとし、またスラック値より最適化

目標値を求める(508)。すべてのクリティカルパスを被最適化パスに分けその最適化目標値が求まったら(509)本処理を中止し(510)、論理合成ヘデータをインターフェースする。すなわち処理402へ進む。

つぎに第4図のステップ403の被最適化パス上の抽象機能素子の選択手順について説明する。

まず、(1)外部出力に最も近い抽象機能素子を捜し出す。これは該抽象素子の変換によって発生する他の回路への影響を最小限にするためである。

(2)最も小さいファンアウトを有する抽象機能素子を捜し出す。これは、いまファンアウトが多い抽象素子を同一機能のより小さい遅延段数をもつ抽象素子に変換したとき、該変換後の抽象素子をノードとして持つパスが最小段数違反パスとなる可能性があるからである。

(3)最も小さいファンインを有する抽象機能素子を捜し出す。これは、いまファンインが多い抽象素子を同一機能のより小さい遅延段数をもつ抽

— 21 —

—612—

— 22 —

象素子に変換したとき、該変換後の抽象素子をノードとして持つパスが最小段数違反パスとなる可能性があるからである。

前記(1)～(3)の優先度は(2)と(3)を満たすものでかつ(1)を満足するものが最優先で、(2)と(3)に関しては(2)の値のより小さい抽象機能素子を選択する。この優先度についてはルールベース方式を用いているので自由に変更できる。

ここで第6図を用いて第1図の論理合成部106における抽象機能素子の展開の実施例について説明する。

第6図(a)に示す10ビット2入力加算器600の展開の手順について説明する。第6図(b)は、2ビット加算器を5つ(601～605)を用いてリップルキャリー10ビット2入力加算器を構成した図である。第6図(c)は2ビット加算器1つ(606)と4ビット加算器2つ(607、608)を用いて10ビット2入力加算器を構成した図である。2ビット加算器のゲート数は20

— 23 —

子への変換について説明する。

いま8ビットの1加算器(インクリメント)を例に取って説明する。レジスタ転送レベル言語で記述された8ビットの1加算器は、INC701という抽象素子にトランスレートされる。そこで論理構成の処理が開始されるとINC701という素子はEXOR(排他的論理和)702、AND703、NOT704の論理素子に展開される。いま第7図(b)の論理素子からなる回路と、第7図(c)の回路と2つが考えられ、第7図(b)の回路では各ビットのキャリーの計算を独自の回路で行っているのに対して、第7図(c)の回路は4ビットの1加算器を2つ用いた構成であり、したがって下位4ビットからのキャリーの計算結果を用いるようになっている。2つの展開回路の違いはそれぞれ他の物と比較して、第7図(b)では信号伝播遅延が小さいのに対して(ゲート数48、ゲート段数4段)、第7図(c)ではゲート数が小さい(ゲート数40、ゲート段数5段)という特徴がある。この展開の処理をルールベ

— 25 —

でゲート段数は3段である。一方4ビット加算器のゲート数は45で段数は5段である。そこで第6図(b)の構造にするとゲート数は100でゲート段数は15段になる。一方、第6図(c)の構造にするとゲート数は110となりゲート段数は12段となる。

いま、クリティカルパス上の10ビット加算器を高速回路に展開する要求が出されているときは、前記メタルールを用いて第6図(c)の回路を選択しこれに展開する。また反対に10ビット加算器がクリティカルパス上にない場合は、メタルールを用いて第6図(b)の回路を選択してこれに展開する。

展開後の抽象機能素子の最適化については、いま第6図(b)の様な回路がクリティカルパス上に存在する場合、2ビット加算器601と602の2つを4ビット加算器1つに置き換えることによって段数が6段から5段に減少し1段数分スピードが速くなる。

つぎに第7図を用いて抽象機能素子から論理素

— 24 —

スで行う場合、まず2つの基本展開ルールが用意されており、前記クリティカルパス探索手段で発見されたクリティカルパス上にINCと言う素子を使用されており、遅延最小の条件で展開の要求が出された場合は、メタルールでそれを判断して第7図(b)の回路に展開し、さもないければ第7図(c)の回路に展開する。

さらに簡単化ルールとして4入力ANDと2入力ANDがシリアルに接続されている場合、3入力AND2つに変換するというものがある場合、上記第7図(b)の回路は第7図(c)の回路になることが考えられるが、ゲート数最小化のために設けられたANDのシリアル結合は簡単化しないようにフラグをつけておく。そうすることによって、遅延時間最小化制約を満たす構造回路を生成できる。

上記実施例は、遅延時間最小化のルールによって生成された回路が、上記簡単化ルールのようなゲート数最小化のルールによって回路が変更されないようにした例であるが、反対にゲート数最小

— 26 —



化のルールによって回路を小さくする場合はフラグを off にすればよい。

以下簡単なモチーフを用いてクリティカルパスの探索と合成の処理を説明する。

第8図は合成前の機能ブロック図である。801～805は入力端子である。ここで入力端子801～803は4ビット幅を有している。806, 807はクロック入力端子である。また808は4ビット幅の出力端子である。809は4ビット2入力加算器で、810は4ビット2入力減算器である。811は比較器で、812は4ビットの1加算器(INC)、813は4ビットの1減算器(DEC)である。814と815はそれぞれ1ビット、4ビットのレジスタで、816と817は2-1のマルチプレクサである。さらに818はANDで、819, 820はインバータである。また821は内部信号である。

まずこの回路の機能素子及び結線情報をスキマテック入力または機能設計記述言語で書く。この情報をデータ変換手段を用いて回路設計データに

変換する。いまこの回路全体を論理合成の対象として遅延時間の制約を与える。ここでは、レジスタR2(815)を終点とするパスに対して最大遅延制約段数10を与える。また出力端子G(808)に最大遅延制約段数19を設定する。この設定の仕方はスキマテック入力ではレジスタシンボルR2のデータ入力端子に対して最大遅延時間制約のテキスト入力テーブルを開いて行うことができる。出力端子Gについても同様である。スキマテックの情報がなくともこれらの制約はコマンドで入力できる。

つぎに各抽象機能素子に対する遅延情報の与え方について説明する。設計者は機能ブロックの遅延時間を自由に選択または設定できることが重要だが、機能ブロックをスキマテック入力するたびに、もしくは機能素子を記述するたびに遅延時間を設定していたのでは設計効率が大変悪いので、自動割当の手段を用いる。抽象機能素子の遅延時間は入力数及びそのビット数に依存し、さらに同一機能に対して構造回路は複数考えられるため、

— 27 —

ここではゲート数も遅延時間も良好な標準的な論理回路を想定して遅延時間を与える。いま第8図の点線の円で囲まれた数字は各機能ブロックの最大遅延段数である。インバータは他の論理ゲートに比べて遅延時間が小さいのでここではゲート段数を0としている。

これを整理して示すと以下の表1のようになる。

表 1

機能ブロックタイプ	遅延段数
A D D	6
S U B	7
C O M P	5
M U X	1
I N C	3
D E C	3
R E G	2
A N D	1
I N V	0

これらは設計データベースから各抽象機能素子のタイプと入力数と入力数のビット幅を取り出し

— 29 —

— 28 —

て、この情報をもとに計算して割り当てたものであり、これらについては合成途中で再度生成される可能性があるためワーク記憶領域に保存しておく。

そこで上で設定した遅延時間制約に基づいたクリティカルパス探索の処理について説明する。

入力端子からの各抽象機能素子の出力における段数を求める。入力点A(802)、B(803)、C(804)、D(805)は0段で、クロック入力CK1(806)にはA～D入力に対して遅延時間として3段遅延を持たせ、CK2(807)は16段とする。

いま入力端子A(802)、B(803)からのデータパスの段数について説明する。点線の四角形の上から1段目の値が入力からの段数を示す。減算器(810)の出力は7段となり加算器(809)の出力は6段となる。またMUX(816)の出力は偽入力(810)からの段数が大きいのでそれをとり8段となる。一方MUX(816)の制御入力CK1(806)が3段でレジスタ

— 30 —

R 1 (814) の段数 2 を加えて 5 段となる。また M U X (816) の出力は最大 6 段となる。従ってこのバスはデータバスからの 8 段をとる。比較器 (811) の出力は F (821) からのバスの 8 段と比較器本体が有している 5 段を加えた 13 段となる。さらに M U X (817) の出力はデータバス F (821) や 1 加算器 (812) の出力段数の 3 段よりも大きい 14 段の値をとる。最終的に、レジスタ R 2 (815) のデータ入力へのバスの最大段数は 14 段となる。

つぎに R 2 (815) を終点とするバスに対して制約段数 (点数の四角形の 2 段目の値) を計算する。R 2 には 10 段の最大遅延の制約が割り付けられている。R 2 へ入力するバスは M U X (817) から来ている。M U X の右から入力する制御信号 822 は偽のときの条件を表しており、823 は真のときの条件を表している。M U X (817) の左から入力する信号はデータ信号で右から入力する制御信号に対応するものである。いま M U X のデータ入力 F (821) に於ける制約を

— 31 —

制御入力に対してレジスタ R 1 (814) の出力は 3 段となり、レジスタ R 1 (814) のクロック入力からデータ出力までの遅延段数を差し引いてクロック入力 C K 1 (806) は 1 段となる。

ここでスラック (制約段数 - 始点からの段数) を求める。点線の 3 段組の長方形の 3 段目の値がスラック値である。そして R 2 (815) からこのスラックの値の内マイナス値をトレースすればこれがクリティカルバスとなる。このモチーフでは -4 のバスが最も危険なバスで、つぎが -3 のバスである。

同様に出力 G (808) については、まず入力 C K 2 (807) からの段数は入力 A (802)、B (803)、E (801) に対して 16 段の遅延を持っているので、レジスタ R 2 (815) のクロック入力では 16 段となり、R 2 のデータ出力では 18 段となる。さらに D E C (813) の出力端子で 21 段となり結局出力 G (808) で 21 段になる。つぎに出力 G には制約 19 段が与えられているので、D E C (813) の入力 D

— 33 —

求めると 9 段になる。ここで F はファンアウトが 2 以上であるので F の出力のものの遅延段数決めることができない。そこで M U X (817) の真入力のデータバスの制約を調べると、偽の場合と同様に 9 段となり、さらに I N C (812) は 3 段なので I N C の入力 D は 6 段となる。このバスについては、いま終点を指定したレジスタの出力に達したのでここで終了する。そこで M U X (817) の制御信号のバスの制約段数を求める。M U X の制御入力端子はそれぞれ 9 段である。M U X の偽のバスのインバートによる遅延は考慮せず比較器 (811) の出力端子に於ける段数は 9 段となる。そして比較器の入力端子に於ける段数を計算する。比較器の遅延段数は 5 段であるので入力 F (821) の出力は先ほど計算した 9 段よりも小さい、いま求めた 4 段を制約とする。以下同様に加算器 (809)、減算器 (810) の出力はそれぞれ 3 段となり減算器 (810) の遅延段数が 7 段であるので A (802)、B (803) はそれぞれ -4 段となる。一方 M U X (816) の

— 32 —

E C 自体が有している 3 段の遅延段数を差し引き 16 段となる。またレジスタ R 2 のクロック入力からデータ出力までの遅延段数を差し引いてクロック入力 C K 2 (807) は 14 段となる。スラックは -2 となりクリティカルバスは C K 2 - R 2 - D E C - G のバスとなる。

つぎに最適化バスの選択方法について説明する。クリティカルバスが有しているスラック値から最も危険度の高いバスを選ぶ。いま R 2 のデータ入力に与えた制約に対して危険度の高いバスから示す。c p 1. 1 と c p 1. 2 はスラックが -4 のクリティカルバスで、c p 2. 1 と c p 2. 2 はスラックが -3 のクリティカルバスで、c p 3. 1 は -2 のクリティカルバスである。個々のクリティカルバスの最適化目標値はスラック値の絶対値とし t a r g e t 1 ~ 3 に示す。以上を総合すると次の第 2 表のようになる。

(以下余白)

— 34 —

第 2 表

```

c p 1 . 1 : A - S U B - M U X - F - C O M P
- M U X - R 2
c p 1 . 2 : B - S U B - M U X - F - C O M P
- M U X - R 2
s l a c k 1 : - 4
t a r g e t 1 : 4
c p 2 . 1 : A - A D D - M U X - F - C O M P
- M U X - R 2
c p 2 . 2 : B - A D D - M U X - F - C O M P
- M U X - R 2
s l a c k 2 : - 3
t a r g e t 2 : 3
c p 3 . 1 : C K 1 - R 1 - M U X - F - C O M
P - M U X - R 2
s l a c k 3 : - 2
t a r g e t 3 : 2

```

いま、c p 1 . 1 に着目するとこれは他の全てのクリティカルパスに重複する部分を有しており、

— 3 5 —

o p 1 : 0

但し o p p 1 は o v p 1 で最適化が実現できた場合にのみ最適化目標値はゼロとなる。

つぎに被最適化パス o v p 1 について適用する最適化手法について説明する。

いま説明する手法は被最適化パス上の複数の機能素子に対して応分の最適化目標値を分担するものである。一方被最適化パス上のある機能素子に対して最大限の最適化目標値を担わせ、最小回数の展開で目標値を実現する手法でも実現できる。

まずはじめに、よりファンアウトの小さい抽象機能素子を選択するがその対象となるものは M U X 8 1 7 である。そこで M U X の展開方法について高速化ルールが存在するかどうか調べ、いま高速化回路に展開するルールが存在しないので（本実施例では存在しないとした）つぎの候補を探す。またファンアウトの小さい抽象機能素子を選択すると C O M P ( 8 1 1 ) が候補に挙がる。そこで C O M P を展開したあとで高速化されるかどうか調べると是であるので実際に展開を行う。展開後

— 3 7 —

これを重複パス o v p 1 とする。最適化目標値は全クリティカルパスのうちもっとも危険度の小さいスラックの絶対値を取るので 2 ( o t 1 ) となる。また c p 1 . 1 は c p 1 . 2 とともに重複しておりこれを o v p 2 と示す。最適化目標値は t a r g e t 1 - o t 1 で 2 ( o t 2 ) となる。

```

o v p 1 : M U X - F - C O M P - M U X - R 2
o t 1 : 2
o v p 2 : S U B - M U X
o t 2 : 2

```

つぎに c p 2 . 1 に着目すると重複パス o v p 1 について処理済みなので、c p 2 . 2 との重複パス o v p 3 を被最適化パスとし、最適化目標値 o v 3 は 1 ( t a r g e t 2 - o t 1 ) となる。c p 3 . 1 については o p p 1 が被最適化パスの候補になるが最適化目標値 o p 1 は 0 ( t a r g e t 3 - o t 1 ) となるため候補からはずす。

```

o v p 3 : A D D - M U X
o v 3 : 1
o p p 1 : C K 1 - R 1 - M U X

```

— 3 6 —

の回路を第 9 図に示す。そして展開後の個々の機能素子に対して遅延段数を与える。展開後の部分回路の段数計算を行うと 4 段になるので 1 段分だけ最適化目標値が達成された。ただし E X O R ( 9 0 1 ) は 2 段、4 入力 N O R ( 9 0 2 ) は 2 段としている。

ところが被最適化パス o v p 1 は最適化目標値が 2 段であるので、まだ上記展開のみでは達成されないためつぎの候補を捜しにいくが、M U X ( 8 1 6 ) も上記理由により最適化展開が不可能である。そのため C O M P の展開後の回路について機能素子間で遅延段数の最適化を行うが、第 9 図のように最適化の余地が残っていないのでつぎの被最適化パスの展開を行う。

被最適化パス o v p 2 について最適化目標値を計算する。重複したクリティカルパス o v p 1 の最適化目標値が実現できなかったのも、本最適化パスはその分最適化目標値が大きくなる。本来の最適化目標値は 2 段であるが、いま o v p 1 の非実現最適化目標値 1 段を加えて 3 段になる。そこで最適化展開候補を捜すと S U B ( 8 1 0 ) が選

— 3 8 —

択され、さらに高速化回路に展開するルールがあるかどうかを調べると存在するので、SUBを高速化回路に展開する。ここでは5段の遅延段数を持つ回路に展開できるので(第10図を参照、1001は4ビット全加算器で遅延段数は5段)、最適化目標値3に対して2段分を満足させる。ovp2には他に最適化展開可能な機能素子がないので、展開後の回路について機能素子間の最適化を行うが、最適化の余地が残っていないのでクリティカルパスcp1.1とcp1.2は最適化目標値4を達成できず3段分しか稼げなかったことになる。同様にovp3の最適化目標値は2段になりパス上のADDは展開後最小5段にしかならないのでクリティカルパスcp2.1、cp2.2の最適化目標値の3段は実現できず1段分の危険度は残る。さらに重複クリティカルパスの最適化目標値が実現できなかったことから、パスopp1は最適化目標値1段を持った被最適化パスになるが、opp1上にはMUXとかREG等最適化候補がないためなにも行われない。従ってクリ

— 39 —

で、論理式の簡単化やハードウェアのインプリメンテーションによる遅延時間の最適化だけを用いた場合よりも遅延時間の制約を満足する回路が実現できるという効果がある。

#### 4. 図面の簡単な説明

第1図は本発明の一実施例にかかる論理回路の合成装置の構成を説明するブロック図、

第2図は本発明の一実施例における論理合成の処理フローを説明する図、

第3図は本発明の一実施例におけるクリティカルパス探索手段の処理を説明する図、

第4図は本発明の一実施例における被最適化回路(被最適化パス)の抽出と該パス上の抽象機能素子の選択、展開及び最適化処理の手順について説明する図、

第5図は本発明の一実施例におけるクリティカルパスから被最適化パスの抽出手法について説明する図、

第6図は本発明の一実施例における抽象機能素子の展開について説明する図、

— 41 —

ティカルパスcp3.1について1段分の危険度が残ってしまう。また各被最適化パスの端点同士の最適化が可能かどうか試みるが本実施例では被最適化パスがMUX(816)で分離されているため不可能である。本実施例によると最長遅延段数が14段から11段に削減された。

本実施例は展開後の回路の遅延段数を既知として説明したが、未知の場合は展開後の回路の段数計算を行う必要がある。さらに展開後の回路同士で遅延段数の最適化が行われた場合は再度クリティカルパス探索手段で展開前のクリティカルパスが回避されたかどうか調べ、クリティカルパス上の機能素子がすべて展開されるまで繰り返す。

#### [発明の効果]

以上説明したように、本発明の装置によれば少ないメモリ容量で抽象回路のレベルでタイミング検証が行え、合成前にタイミング仕様が満たされるか否かの判定が可能になる。また、抽象回路に割り当てられた遅延時間を利用して、回路の展開及び最適化時にスピードの速い回路を生成するの

— 40 —

第7図は本発明の一実施例における抽象機能素子から論理素子への変換について説明する図、

第8図は本発明の一実施例におけるクリティカルパスの探索と合成の処理を説明するのに使用する簡単なモチーフの機能ブロック図、

第9図は本発明の一実施例における比較器の論理素子への変換について説明する図、および

第10図は本発明の一実施例における減算器の展開について説明する図である。

101…レジスタ転送仕様の入力部

104…タイミング制約の入力部

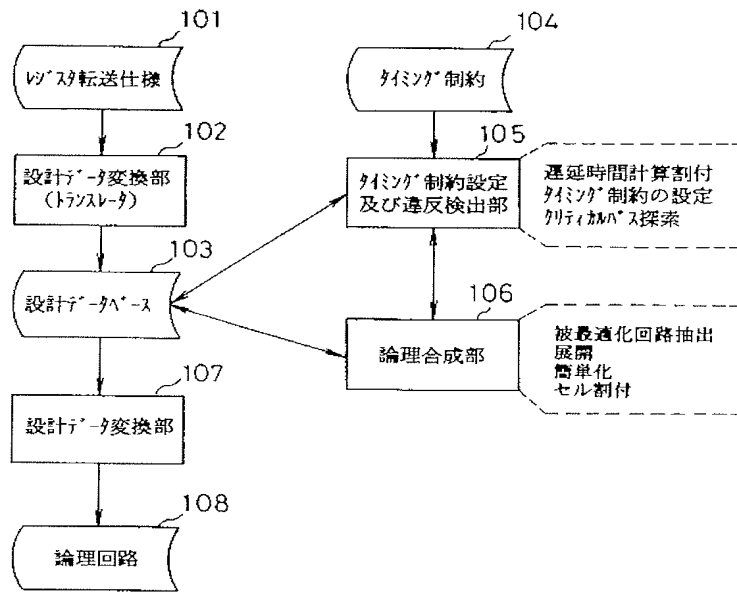
105…タイミング制約設定及び違反検出部

106…論理合成部

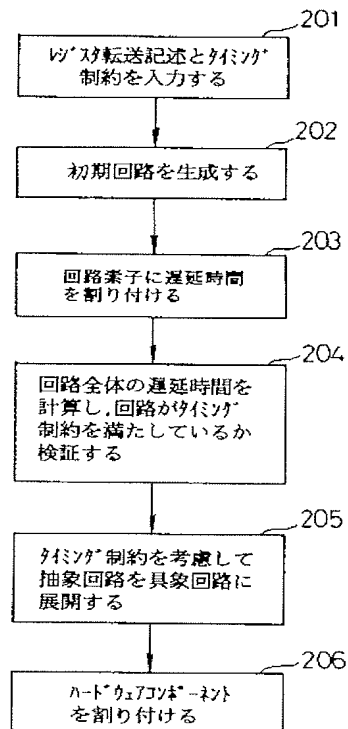
108…論理回路の出力部

代理人弁護士 三好秀和

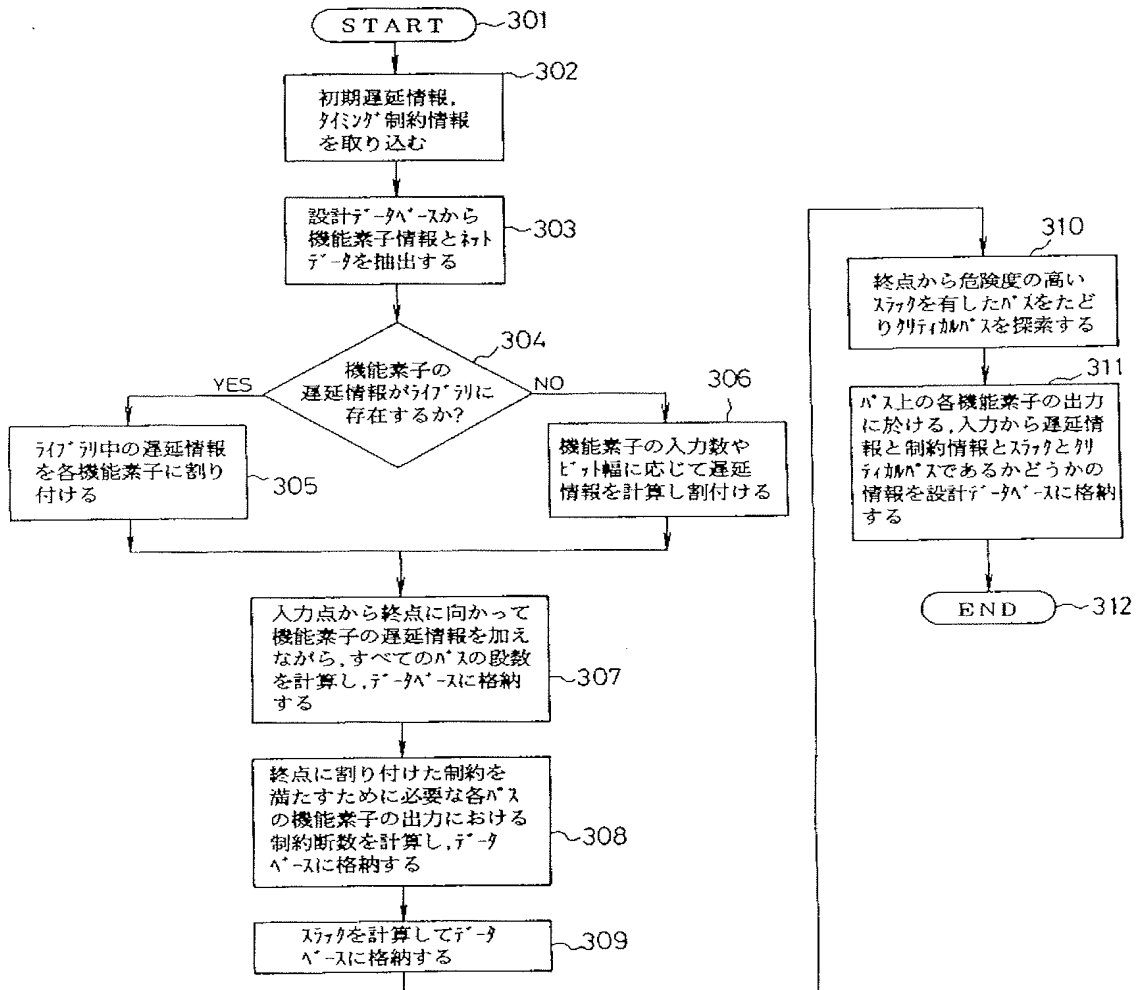
— 42 —



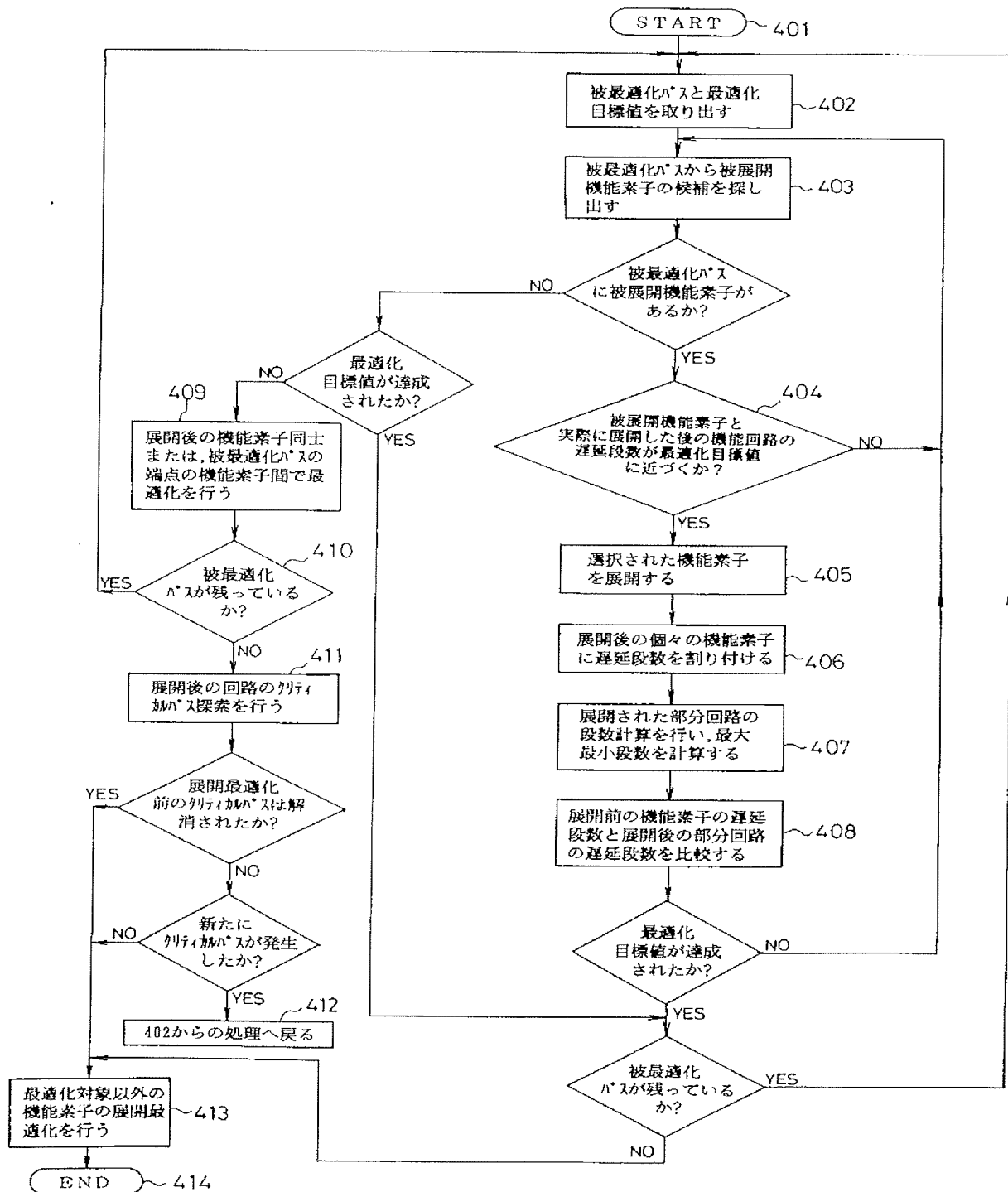
第 1 図



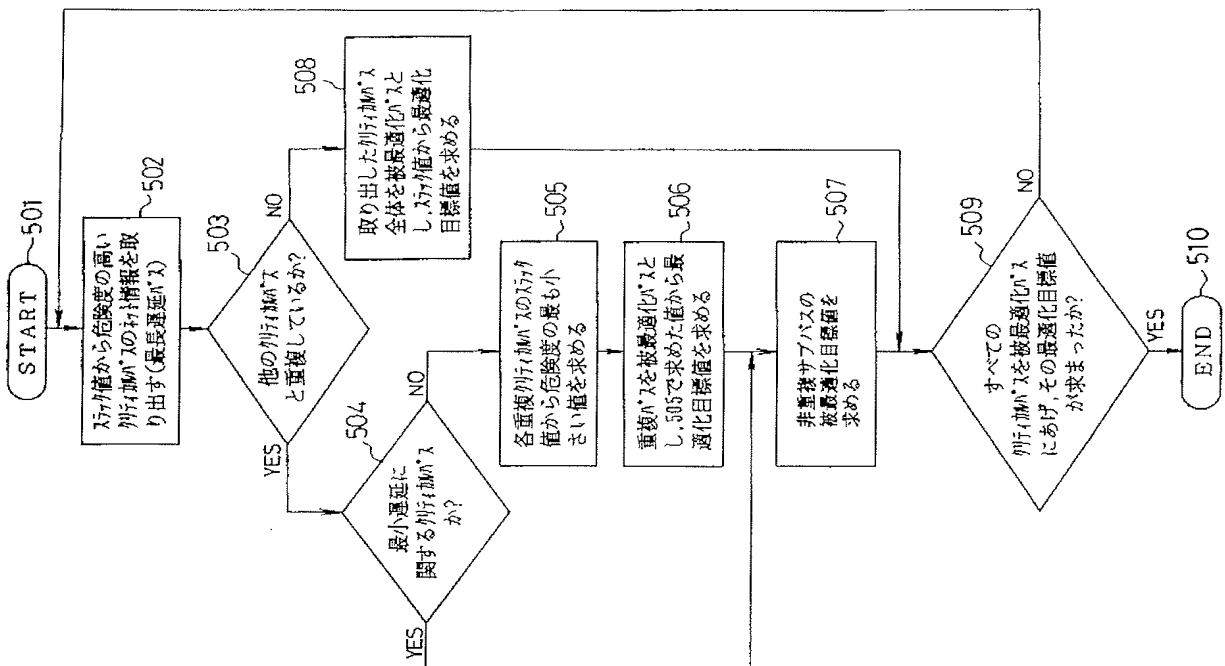
第 2 図



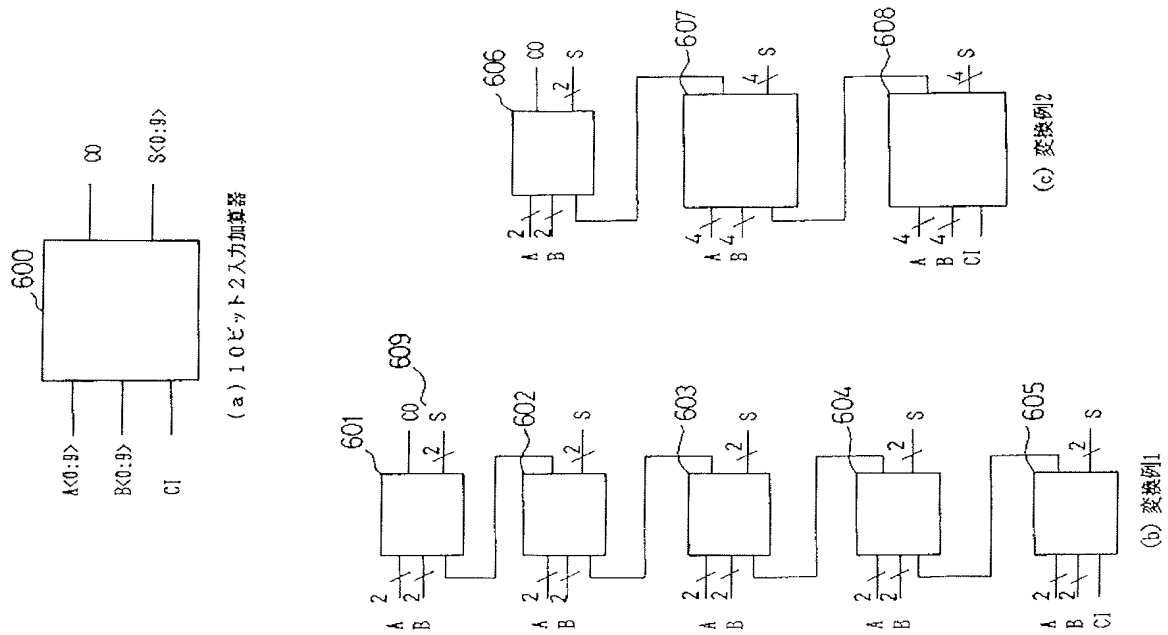
第 3 図



第 4 図

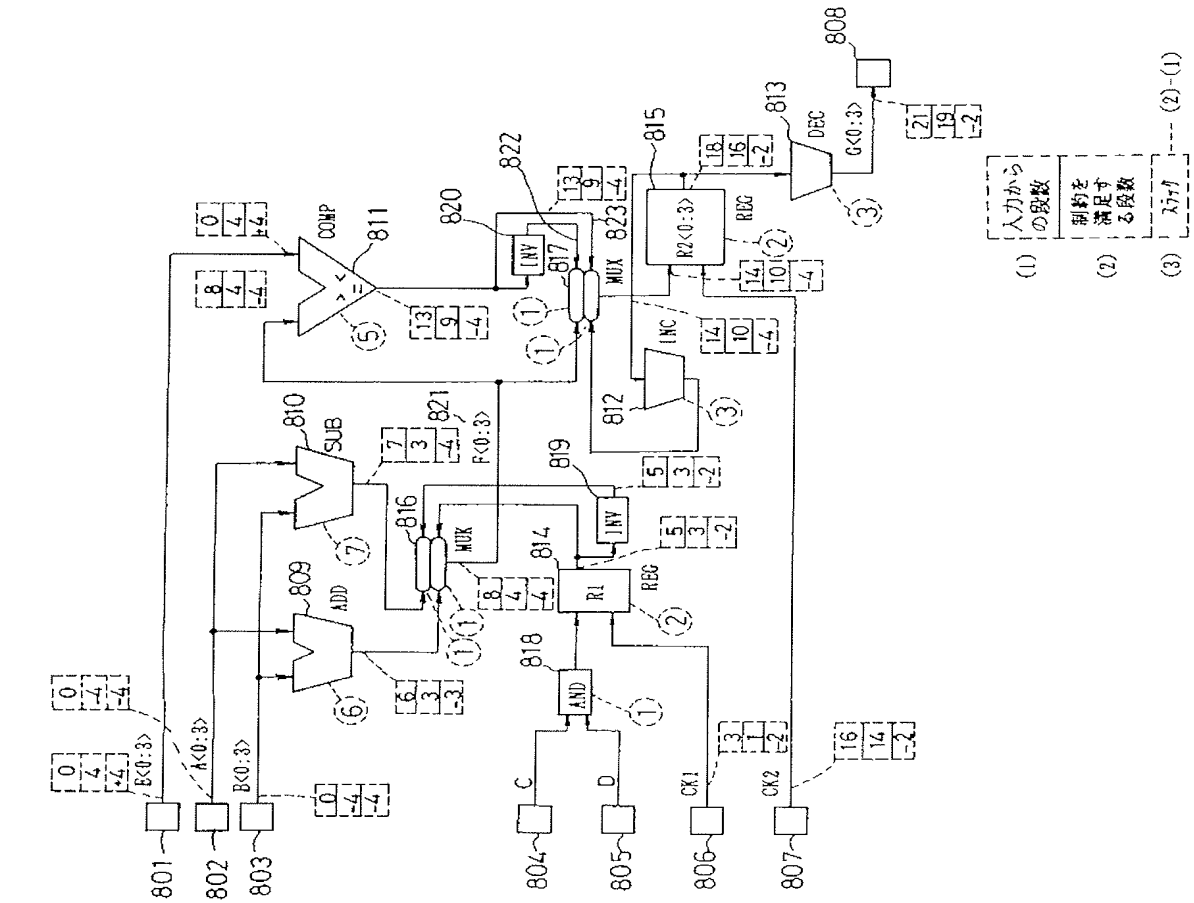


第 5 図

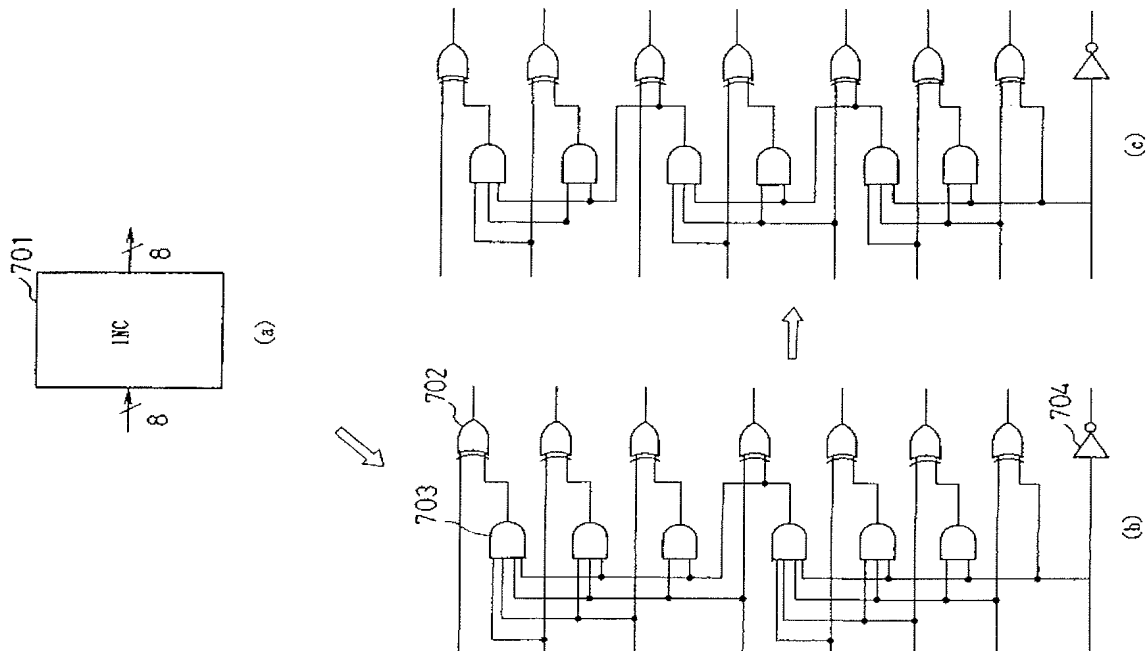


第 6 図

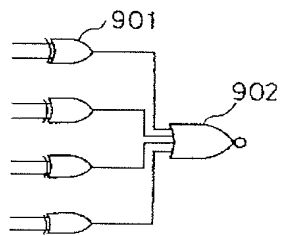




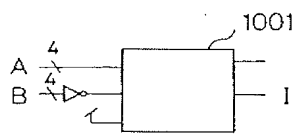
第 8 図



第 7 図



第 9 図



第10 図